Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2011-023 April 16, 2011

Gasping for AIR Why we need Linked Rules and Justifications on the Semantic Web

Lalana Kagal, Ian Jacobi, and Ankesh Khandelwal

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Gasping for AIR — Why we need Linked Rules and Justifications on the Semantic Web *

Lalana Kagal
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Lab
lkagal@csail.mit.edu

Ian Jacobi
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Lab
pipian@mit.edu

Ankesh Khandelwal
Rensselaer Polytechnic Institute
Tetherless World Constellation
ankesh@cs.rpi.edu

ABSTRACT

The Semantic Web is a distributed model for publishing, utilizing and extending structured information using Web protocols. One of the main goals of this technology is to automate the retrieval and integration of data and to enable the inference of interesting results. This automation requires logics and rule languages that make inferences, choose courses of action, and answer questions. The openness of the Web, however, leads to several issues including the handling of inconsistencies, integration of diverse information, and the determination of the quality and trustworthiness of the data. AIR is a Semantic Web-based rule language that provides this functionality while focusing on generating and tracking explanations for its inferences and actions as well as conforming to Linked Data principles. AIR supports ***Linked Rules***, which allow rules to be combined, re-used and extended in a manner similar to Linked Data. Additionally, AIR explanations themselves are Semantic Web data so they can be used for further reasoning. In this paper we present an overview of AIR, discuss its potential as a Web rule language by providing examples of how its features can be leveraged for different inference requirements, and describe how justifications are represented and generated.

1. INTRODUCTION

Though RDF Schema (RDFS) and the Web Ontology Language (OWL 1 & 2) provide some reasoning capability over Resource Description Framework (RDF) data, the application of Semantic Web technologies to e-government, business, policy management, workflow systems, and many other fields requires more expressive rule languages to capture the underlying system logic. Much of the Semantic Web's growth today has been in the form of Linked Data, which means that any Web rule language must be able to handle highly interconnected and spatially dispersed data. It must be able to dynamically traverse this web of data to find additional facts to support the conclusions of its reasoner. Furthermore, a Web rule language should expose its rules as Linked Data as well so that they can be re-used and combined in a similar manner.

Web rule languages must also be able to cope with the problems that arise from the inherent openness of the Web. Reasoning over data on the Web can easily lead to logical inconsistencies as anyone can assert anything. For example, a reasoner could infer multiple subjects for the same value of an **Inverse Functional** property, **foaf:mbox_sha1sum**[1], caused by the incorrect copying of someone's Friend of A Friend (FOAF) page, leading to a logical inconsistency. A Web rule language must be able to isolate the results of reasoning [20] to prevent them from causing inconsistencies in the global state.

Another problem with Web systems involves the trustworthiness of data — what data may be trusted and what criteria may be used for a decision. Different trust levels may be assigned to Web documents and the facts contained therein. For example, a hospital may be trusted with information about a potential virus outbreak but may not be trusted with respect to its economic inflation predictions. The ability to access only trusted RDF subgraphs from Web pages is useful in maintaining the quality of inference results.

The quality of these results also depends on the provenance of the data as well as other rules used to make inferences. In order to evaluate deductions made by others, deduction traces, or justifications as they are known, are also required [8, 9]. They provide detailed provenance information, including the data sources and rules applied, to allow applications to evaluate the trustworthiness of a particular result through automated proof checking [3, 15]. Capturing and tracking this justification information is another important property of Web rule languages.

AIR (**A**ccountability **I**n **R**DF) is a Semantic Web rule language that provides for the isolation of reasoning and management of trust while emphasizing justifications and ***Linked Rules***. ***Linked Rules*** conform to Linked Data principles[2] and enable AIR rules to be combined, re-used

*This material is based upon work supported by the National Science Foundation under Award No. CNS-0831442, by the Air Force Office of Scientific Research under Award No. FA9550-09-1-0152, and by Intelligence Advanced Research Projects Activity under Award No. FA8750-07-2-0031.

[1] `http://xmlns.com/foaf/spec/#term_mbox_sha1sum`
[2] Linked Data Design Issues, http://www.w3.org/DesignIssues/LinkedData.html

and extended similar to Linked Data. AIR is represented in Notation 3 (N3)[3], a superset of RDF which includes variable quantification and graph quoting. AIR extends these features to provide named rules, functions to selectively query SPARQL Query Language for RDF (SPARQL) endpoints and perform contextualized scoped reasoning. It also supports functions for retrieving subgraphs from Web resources and basic cryptographic, string, and math operations. In addition to returning any deduced triples, the AIR reasoner returns a justification for these deductions. AIR justifications are also in N3, making it straightforward to define AIR rules that reason over the inferences and justifications of other AIR rules [10].

AIR has been used in various projects to meet different rule-based inferencing requirements. It has been used for controlled information exchange between government agencies where decisions made by external rules were frequently required making contextualized reasoning very important [23]. It has been used to secure access to Web resources [17] and SPARQL endpoints [6] based on the credentials of the user. AIR has also been used to analyze database queries [11] with respect to privacy policies that required ontology-based reasoning, querying of large knowledge bases for factual information and expressive non-monotonic reasoning. Lastly, AIR has been used in accountability mechanisms for processing audit logs and looking for data usage outside of what was allowed by usage restriction policies [24]. All of the above projects made use of justifications for debugging and accountability purposes.

AIR, as described in [9], was initially proposed as a policy language that used truth maintenance to generate justifications. In this paper, we explore and extend the language with unique capabilities required by ***Linked Rules***, define a new justification ontology that captures the AIR reasoning process to allow proof checking and further reasoning, and broaden the functionality of AIR into a general Web rule language.

This paper is structured as follows: we start by describing a motivating scenario in Section 2. We then discuss the concept of Linked Rules in more detail in Section 3 before providing an overview of the AIR language in Section 4. In section 5, we discuss AIR's support for justifications — how they are generated and the representation schema. This is followed by a comparison to related work in Section 6. We conclude the paper with a summary and directions for future work in Section 7.

2. MOTIVATING USE CASE

To better understand how AIR's features may be used to solve complex problems, consider the following scenario:

Alice, Bob, and Carol are graduate students who all attended the ESWC conference in 2010. Bob, being an amateur photographer, decided to bring along a camera and take pictures of the conference. When he got home, Bob decided to post all of the pictures he had taken on MITBook, a decentralized social network that Alice, Bob, and Carol are all members of, which features advanced privacy controls. MITBook uses FOAF+SSL[4], a decentralization authentication mechanism that associates FOAF pages with users.

Alice is rather picky about who can see her personal pictures. Bob, knowing that Alice would probably not like the rather incriminating photos taken of her at a dinner following the conference to become widespread, would prefer that his pictures that feature Alice abide by *her* preferred image policy, even if those preferences change.

Bob's co-worker Carol, on the other hand, maintains a meticulously detailed ruleset that ultimately calculates a degree of trustworthiness of every person she knows, so that she can selectively reveal information only to certain trusted people. Bob, knowing the care that Carol puts into this metric, would like to use it for the people Carol knows, but would prefer to not actually use Carol's rules about which photos should be shown to them. Instead, he would like to ensure that only people with a trust-rating of more than 70 can see the photos he posts.

For all other pictures taken at the ESWC conference, Bob would prefer to share them only with attendees of ESWC from MIT and RPI, where his friends are located. As information about all attendees of ESWC and their affiliation are available on a SPARQL server, Bob would like to query that SPARQL endpoint[5] to determine whether an individual was from MIT or RPI, rather than relying on loading the entire dataset before reasoning.

These example policies, resultant inferences and justifications are available at `http://dig.csail.mit.edu/2011/Papers/WWW-AIR/example/`.

3. LINKED RULES

Most rules, whether laws, security policies, business rules, or workflow plans, are rarely defined by a single entity or exist in a single document. They usually comprise of several interdependent rules that are defined and maintained by different entities. Additionally, rules may reference external rules, including those of other organizations. Consider Florida's Sunshine Law, which sets out the rules by which the public may access the public records of governmental bodies in Florida.[6] As seen in Figure 1, the Sunshine Law may be modelled as a series of rules and exceptions that link to external rules of custodians of requested records as well as those of law enforcement agencies that might be using those records. AIR models this kind of rule re-use and linkage through:

- Rule names: AIR rules can be uniquely identified by Uniform Resource Identifiers (URIs) and are accessible according to the principles of Linked Open Data. This allows AIR rules to be treated as first-class objects and enables them to be spatially dispersed but combined during reasoning. Annotations about additional properties of rules such as trust, provenance and validity period can be made and policies which restrict access to and the usage of the rules themselves can be specified.

- Re-use: Rules may be developed modularly by combining, re-using and extending existing rules. Rule specialization is possible where additional conditions or effects can be added to existing rules. In our use case, Bob wants to use Alice's policy for pictures that contain her. He may do this by adding a condition

[3]Notation 3 (N3), http://www.w3.org/DesignIssues/Notation3
[4]FOAF+SSL, `http://esw.w3.org/Foaf+ssl`
[5]Semantic Web Dog Food SPARQL endpoint, http://data.semanticweb.org/sparql
[6]http://www.myflsunshine.com/sun.nsf/pages/Law

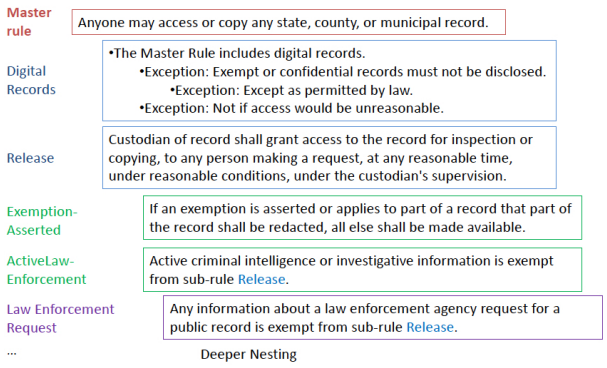

Figure 1: Rule Nesting and Linking in Florida Sunshine Law (119.01(1))

which checks that the picture is of Alice before linking to Alice's policy.

- Scoped re-use: In certain cases, such as when a rule or its creator are not completely trusted or when all inferences of a rule are not of equal quality, executing a rule in its entirety and accepting all its inferences is not feasible. AIR allows for the recursive execution of rules against a certain context and its conclusions to be selectively queried.

This conformance of AIR rules to Linked Data principles forms the basis of ***Linked Rules*** which provides a more natural way to think about and model real world rules, laws and policies on the Web.

3.1 Rule Re-use

One method for rule reuse is currently offered by the Rule Interchange Format (RIF) [14] and Web Ontology Language (OWL), namely that of rule and ontology importing. In doing so, however, all the rules of the imported document become part of the ruleset. In AIR, individual rules can be imported and used as a nested rule. These rules can also be specialized through the addition of conditions or effects.

When rule authors create rules, not all assumptions may be made explicit; often the data to which the rules are applied respect certain implicit assumptions. When these rules are to be used in a different context, these assumptions must be made explicit. In AIR, rules may be reused as "nested rules" that may fire only after "parent rules" which explicitly enforce these implicit assumptions. This allows for the extension or specialization of external rules. Nesting rules in this fashion will necessarily expose the current state of the reasoning (such as currently bound variables) to the external rule being nested. Likewise, the external rule being nested may assert triples which are irrelevant to the policy reusing the rule.

An alternative approach for rule reuse provided by AIR is the use of scoped contextualized reasoning over external rules. This permits external rules to be referenced without giving them access to the current state of bindings or allowing them to assert triples or change the state. Their processing is restricted through the creation of an additional context and scope. Certain trusted inferences can then be extracted and used for reasoning in the original scope.

4. AIR OVERVIEW

AIR is an extension to N3Logic [4] and has been structured to meet the justification and rule reusability requirements of Web information systems. It has a production rule structure[7] that has been found to be useful in expressing different forms of rules. Along with including the N3Logic features of scoped negation, scoped contextualized reasoning, nested graphs, and built-in functions, AIR also supports ***Linked Rules*** and is focused on generating useful ***justifications*** for all actions made by the reasoner. Like N3Logic, AIR is written in N3, which provides a human-readable syntax for a superset of RDF. N3Logic extends the RDF data model by allowing for the quantification of variables as URIs with the **@forAll** and **@forSome** directives. It also permits the inclusion of nested graphs by using curly braces to quote subgraphs.

AIR consists of a set of built-in functions and two independent ontologies — one is for the specification of AIR rules, and another for describing justifications of the inferences made by AIR rules. The built-in functions allow rules to access Web resources, query SPARQL endpoints, and perform scoped contextualized reasoning, as well as basic math, string and cryptographic operations. While developing the rule ontology, we focused on capturing how real world rules and laws are written to allow them to be represented naturally in AIR. For the justification ontology, our focus was on re-usability of justifications and on automated proof checking. When given some AIR rules as input, defined using the AIR rules ontology, and some Semantic Web data, the AIR reasoner produces a set of inferences that are annotated with justifications, described in the justification ontology.

All the examples in the paper are in N3. Please refer to `http://www.w3.org/2000/10/swap/Primer` for an overview of N3 and to Figure 2 for the list of namespaces used in the paper.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix math: <http://www.w3.org/2000/10/swap/math#> .
@prefix str: <http://www.w3.org/2000/10/swap/string#> .
@prefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#> .
@prefix airj:
   <http://dig.csail.mit.edu/2009/AIR/airjustification#> .
@prefix sparql: <http://www.w3.org/2000/10/swap/sparqlCwm#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix sioc: <http://rdfs.org/sioc/ns#> .
@prefix tags:
   <http://www.holygoat.co.uk/owl/redwood/0.1/tags/tags#> .
@prefix swrc: <http://swrc.ontoware.org/ontology#> .
@prefix req:
   <http://dig.csail.mit.edu/2011/Papers/WWW-AIR/example/req#> .
```

Figure 2: Namespaces

[7]Production rules, http://en.wikipedia.org/wiki/Production_system

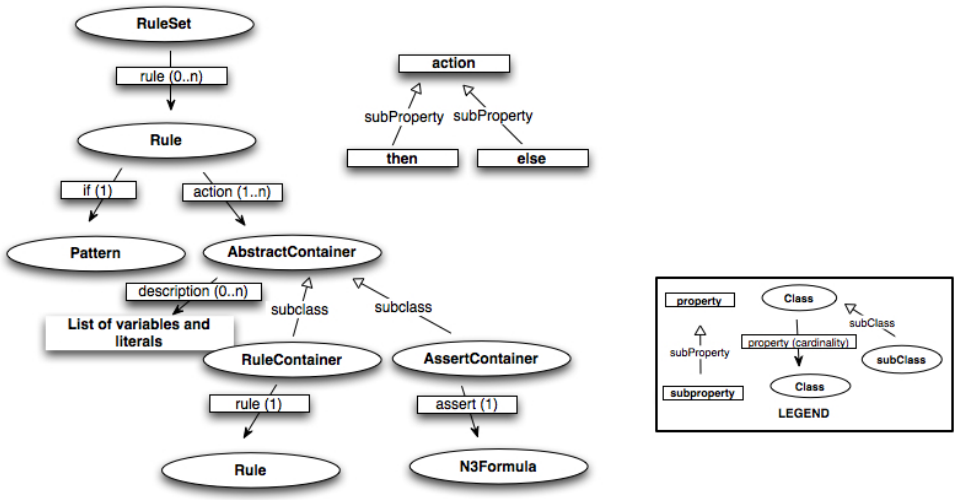

Figure 3: AIR Rule Ontology

4.1 AIR Rules

As illustrated in Figure 3, AIR rules are defined using the following properties: ***air:if***, ***air:then***, ***air:else***, ***air:description***, ***air:rule*** and ***air:assert***. Every rule is named with a URI, and rules are grouped into ***air:RuleSets*** or nested under other rules. This nesting can happen either under the ***air:then*** property or the ***air:else*** property. The rules nested directly under the ***RuleSet*** are referred to as the top rules of the ruleset. A chain of rules is defined as a sequence of rules, such that every rule, barring the first in the chain, is nested under either the ***then*** or the ***else*** of the preceding rule. Figure 4 provides an example of nested rules. In this case, `alice:MyImgPolicy` only becomes active if the ***air:if*** of the parent rule, `:ViewImageRule1`, matches a pattern in the knowledge base.

There are three kinds of rules in AIR — ***air:Belief-rule***, ***air:Hidden-rule*** and ***air:Elided-rule***. All rules are, by default, ***Belief-rules***. The descriptions and conditions of ***Belief-rules*** contribute to the overall justification. `:ViewImageRule1` in Figure 4 is an example of a ***Belief-rule***. In contrast, ***Hidden-rules*** and ***Elided-rules*** are used to modify the default justification. (Please refer to Section 5.2 for more information about justification generation and modification).

The conditions of a rule (for example, **:REQUESTER a req:Requester.** in Figure 4) are defined as graph patterns which are matched against RDF graphs, much like the Basic Graph Pattern (BGP) of SPARQL queries[8]. If the condition matches the current state of the world, defined as the facts known or inferred to be true so far, then all the actions under ***then*** (then-actions) are fired, otherwise all the actions under ***else*** (else-actions) are fired. The condition matches the current state if there is a subgraph of known facts that matches the graph pattern. This subgraph is referred to as the matched graph.

Existentially quantified variables may be declared within graph patterns by using the **@forSome** directive. Any universally quantified variables, quantified using **@forAll**, are declared outside of the rule. The scope of a existentially quantified variable is the graph pattern in which it is declared, whereas that of a universally quantified variable is any chain of nested rules. The URIs that are universally quantified, existentially quantified and those that are not quantified (resources) are assumed to be disjoint. When rules are imported and there is a clash, the behavior is undefined and resolution depends on the implementation. Furthermore, the same URI should be used to share variable bindings with nested rules.

Rules with conditions where some graph pattern must match the current state and others should not match the current state can be specified through the nesting of rules. The actions under ***then*** and ***else*** (together referred to as actions) are defined by an assertion pattern using the ***air:assert*** property, or a rule reference, using the ***air:rule*** property. All actions may be annotated with the natural-language description of the rule or action through the use of the ***air:description*** property that can also contain variables.

When the action is executed, the variables in an assertion pattern (for example, the variable `:REQUESTER` in Figure 4) are substituted with their bindings, and the pattern is asserted. If a rule reference is defined instead, an instance of that rule, created by substituting the variable bindings acquired so far, is activated. The variables in any ***air:description*** property are also instantiated, and the description is maintained by the reasoner. Please refer to Section 4.3 for more information on the order in which rules are fired.

Any asserted graph pattern cannot contain blank nodes or existentially quantified variables. When a rule containing an ***air:else*** property is activated, its condition cannot contain unbounded universally quantified variables.

Since AIR supports ***Linked Rules***, AIR rules may be identified by their URIs which allow them to be easily reused and developed modularly. For instance, `alice:MyImgPolicy` in Figure 4 is more fully defined outside of Bob's rules document, and AIR semantics cause them to be included during the reasoning of their parent rule, `:ViewImageRule1`. The

[8] `http://www.w3.org/TR/rdf-sparql-query/#BasicGraphPatterns`

```
@forAll :REQUESTER, :PIC .

:ViewImageRule1 a air:BeliefRule;
  air:if {:REQUESTER a req:Requester ;
             req:requestedImg :PIC.
          :PIC sioc:topic
            <http://dig.csail.mit.edu/2008/02/rmp/alice-foaf#me>.
   };
   air:then [ air:description (
                  "If the picture requested contains "
                  "Alice then execute Alice's policy "
                  "about image access");
              air:rule alice:MyImgPolicy
   ].

alice:MyImgPolicy
   air:then [ air:description(
                  "Alice's policy has executed");
               air:assert {
                     :REQUESTER req:compliant-with :BobRuleSet }
   ];
   air:else [ air:assert {
               :REQUESTER req:non-compliant-with :BobRuleSet } ].
```

Figure 4: Example AIR RuleSet: Following from the motivating use case, this rule uses Alice's policy to determine whether access to a picture should be granted.

effect of reusing a rule is that all the rule chains starting with that rule are also reused.

Note, however, that we may still extend the actions that fire as a result of matching (or failing to match) `alice:MyImgPolicy`. In Figure 4, we extend the rule to additionally assert that it is (or is not) compliant with `:BobRuleSet` when `alice:MyImgPolicy` completes (or fails) its match.

4.2 AIR Built-ins

AIR supports most N3Logic built-ins including those for cryptographic, math, string, list and time functions. AIR also supports the N3Logic built-ins provided within the ***log:*** namespace. This allows for rules to access Web documents using the built-in function ***log:semantics***, and utilize subgraph matching with the ***log:includes*** property.

As seen in Figure 5, AIR further extends the set of N3Logic built-in functions to support the execution of remote SPARQL queries using ***sparql*** built-ins. SPARQL CONSTRUCT queries may be sent to SPARQL endpoints using the ***sparql:queryEndpoint*** property assertion, and subgraph patterns may retrieved from the graph returned by the endpoint.

The introduction of ***sparql:queryEndpoint*** is unique among rule systems in that it enables the dynamic construction and use of queries against remote knowledge bases rather than relying on a local knowledge base. This is particularly useful as it means that large sets of Linked Data need not be loaded into the knowledge-base prior to reasoning over some rules. Instead, it is possible to write rules that use queries to only extract relevant information from a SPARQL endpoint. This defers the maintenance and querying of this data to external hosts more capable of doing so.

AIR's ability to incorporate the contents of SPARQL queries meets the third requirement of the motivating use case; Bob may only allow access to images from members of MIT and RPI who, according to the SPARQL endpoint, attended ESWC2010. The rule in Figure 5 includes a SPARQL

```
@forAll :REQUESTER, :PIC .

:ViewImageRule3 a air:BeliefRule;
    air:if {
        :REQUESTER a req:Requester.
        :PIC a req:RequestedImg;
             tags:taggedWith [ tags:tagName "ESWC2010" ].
    };
    air:then [ air:description (
                   "If the picture was taken at ESWC2010, "
                   "allow access to MIT and RPI "
                   "organizers");
               air:rule :QueryEndpointForAffiliations
    ] .

:QueryEndpointForAffiliations a air:BeliefRule;
    air:if {
        @forSome :SPARQL, :RESULTS .
        ("""
PREFIX swc:<http://data.semanticweb.org/ns/swc/ontology#> .
PREFIX swrc:<http://swrc.ontoware.org/ontology#> .
PREFIX req:
  <http://dig.csail.mit.edu/2011/Papers/WWW-AIR/example/req#> .
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
CONSTRUCT { """ :REQUESTER """ swrc:affiliation req:MITorRPI .}
WHERE { ?ROLE swc:heldBy ?PERSON.
        ?PERSON rdfs:seeAlso """ :REQUESTER """.
        ?ROLE swc:isRoleAt
            <http://data.semanticweb.org/conference/eswc/2010>.
        {
{ ?PERSON swrc:affiliation
  <http://data.semanticweb.org/organization/rpi> }
UNION
{ ?PERSON swrc:affiliation
  <http://data.semanticweb.org/organization/mit> }
        }
}""") str:concatenation :SPARQL .
        ( <http://data.semanticweb.org/sparql/> :SPARQL )
            sparql:queryEndpoint :RESULTS .
        :RESULTS log:includes {
            @forSome :R .
            :R swrc:affiliation req:MITorRPI .
        }
    } ;
    air:then [ air:description(
                   "The requester," :REQUESTER ", was "
                   "an organizer of ESWC2010 and is a "
                   "member of MIT or RPI, so I will "
                   "allow access");
               air:assert{
                   :REQUESTER req:compliant-with
                       :BobRuleSet } ] .
```

Figure 5: Querying a SPARQL endpoint Within AIR Rules: An AIR rule that may be used to query the SPARQL endpoint to discover if a requester was an attendee of ESWC2010 from RPI or MIT.

CONSTRUCT query to determine the requester's affiliation and extract a graph that may be matched using the ***log:includes*** built-in. If this graph contains a desired triple **:R swrc:affiliation req:MITorRPI .**, it would allow for the conclusion that the requester is actually a member of MIT or RPI who attended ESWC2010.

N3Logic provides scoped contextualized reasoning over N3Logic rules with its ***log:conclusion*** built-in. As the AIR reasoner uses semantics to represent rules that differ from those of N3Logic, we introduce the ***air:justifies*** built-in to check if the execution of some external AIR rules (AIR-closure) against some Semantic Web data produces (RDF-entails) a certain RDF graph. The results of these external rules are not directly included into the current state but can be queried selectively using ***log:includes***. Together, the ***log*** and ***air:justifies*** built-ins provide *scoped contextualized*

```
@forAll :REQUESTER, :PIC .

:ViewImageRule2 a air:BeliefRule;
    air:if {
        @forSome :Person, :S.
        :REQUESTER a req:Requester;
                   req:requestedImg :PIC.
        <http://dig.csail.mit.edu/2008/02/rmp/carol-foaf#i>
            log:semantics :S.
        :S log:includes {
            <http://dig.csail.mit.edu/2008/02/rmp/carol-foaf#i>
                foaf:knows :REQUESTER }.
    };
    air:then [ air:description (
                   "If the requester is a person who "
                   "Carol knows then use Carol's "
                   "inferences about this person's "
                   "trust value");
               air:rule :ScopedExecutionOfCarolRule
    ].

:ScopedExecutionOfCarolRule a air:BeliefRule;
    air:if {
        @forSome :P, :R, :V, :A, :B, :C.
            <http://dig.csail.mit.edu/2011/Papers/WWW-AIR/
                                  example/carol-policy.n3>
            log:semantics :P.
        :REQUESTER log:semantics :R.
        ( ( :R ) ( :P ) ( req:trustvalue ) )
            air:justifies { :B req:trustvalue :V }.
        :V math:greaterThan 70;
    };
    air:then [ air:description(
                   "Carol trusts the requester, "
                   :REQUESTER ", more than 70 percent, "
                   "so I will allow access");
               air:assert{ :REQUESTER req:compliant-with
                           :BobRuleSet}
    ].
```

Figure 6: Recursive Execution of AIR Rules: *air:justifies* may be used to recursively call rule sets, as in this example where `:CarolTrustRuleSet` is executed from within `:ScopedExecutionOfCarolRule`.

reasoning over AIR rules. Figure 6 provides an example of the use of ***air:justifies*** in order to calculate Carol's trust value for a friend of Carol, using rules defined in a separate file, <http://dig.csail.mit.edu/2011/Papers/WWW-AIR/example/carol-policy.n3>. AIR thus meets another need of our use case: that Bob should be able to reuse and execute Carol's rules as defined in other documents.

4.3 AIR Semantics

The procedural semantics of AIR [13] describe how AIR rules fire and how inferences are made. The AIR reasoner applies forward chained reasoning to compute the closure of AIR rules and the input data. When AIR rules fire, their actions, substituted with known variable bindings, are performed. As a result, new rules may be added to the rule base and/or new facts may be deduced. Initially the rule base contains only the top rules in the ruleset, and the fact base is the input facts. The rules in the rule base are said to be active. The active rules whose conditions match the current state of the world (fact base) are referred to as successful rules, whereas those active rules whose conditions have no match are called failed rules.

AIR reasoning is performed in stages. In any given stage, the successful rules are given priority over failed rules and their then-actions are effected before failed rules fire. When all successful rules have fired the world is temporarily closed and the else-actions of all the failed rules are fired simultaneously with the belief that the conditions of all the failed rules are false. AIR reasoning enters the next stage once the failed rules have all fired.

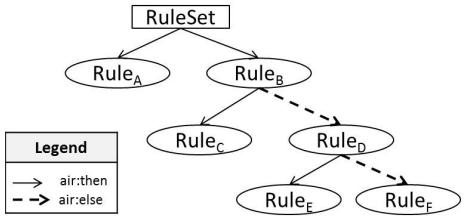

Figure 7: Example of AIR Rule Nesting

In order to illustrate rule nesting, we consider an arbitrary nesting shown in Figure 7. Before entering stage 1, only Rule$_A$ and Rule$_B$ are in the rule base. Then, if the condition of Rule$_B$ is satisfied, Rule$_C$ would be active in stage 1. However, if in stage 1 Rule$_B$ doesn't succeed, then Rule$_D$ will be added to the rule base after the world is closed for stage 1, and will be active from stage 2. Now, in stage 2, if Rule$_D$ succeeds then Rule$_E$ will become active in stage 2. Otherwise Rule$_F$ will be active from stage 3. Note that if Rule$_B$ succeeds in later stages, say stage 3, then Rule$_C$ will also be active (in addition to Rule$_D$) from stage 3 onwards.

The declarative semantics of AIR [13] are defined through translation of AIR rules to stratified Logic Programs [21]. The AIR-closure computation is polynomially-complete in data-complexity and exponentially-complete in program-complexity, where data complexity is the complexity of computing the closure when the program is fixed and the facts are input and program complexity is the complexity when the facts are fixed and program is input.

5. AIR JUSTIFICATIONS

Upon finalizing its reasoning results, the AIR reasoner produces a justification that contains sufficient information to understand the actions made by the reasoner and to debug the rules, if needed. We have developed a justification ontology based on basic Proof Markup Language (PML) concepts [19] to represent the important operations involved in the closure computation as defined by the AIR semantics.

As AIR justifications themselves are N3 data, they can be consumed by other N3 reasoners, including AIR itself, to evaluate the *quality* and *trustworthiness* of the results based on the rules and data sources used. If rules, data, or their creators have belief or trust values associated with them, AIR rules can be written to trace through justifications to evaluate the inferences based on these values.

5.1 Justification Ontology

The AIR justification ontology extends certain PML [19] concepts as shown in Figure 8. PML is a general proof language or "proof interlingua" that is used to describe proof steps generated by different kinds of reasoning engines. We use the PML-Lite vocabulary[9] that represents a subset of PML and is modeled as Events, which can be conveniently used for representing the AIR rea-

[9]PML-Lite - http://tw.rpi.edu/proj/tami/PML-Lite

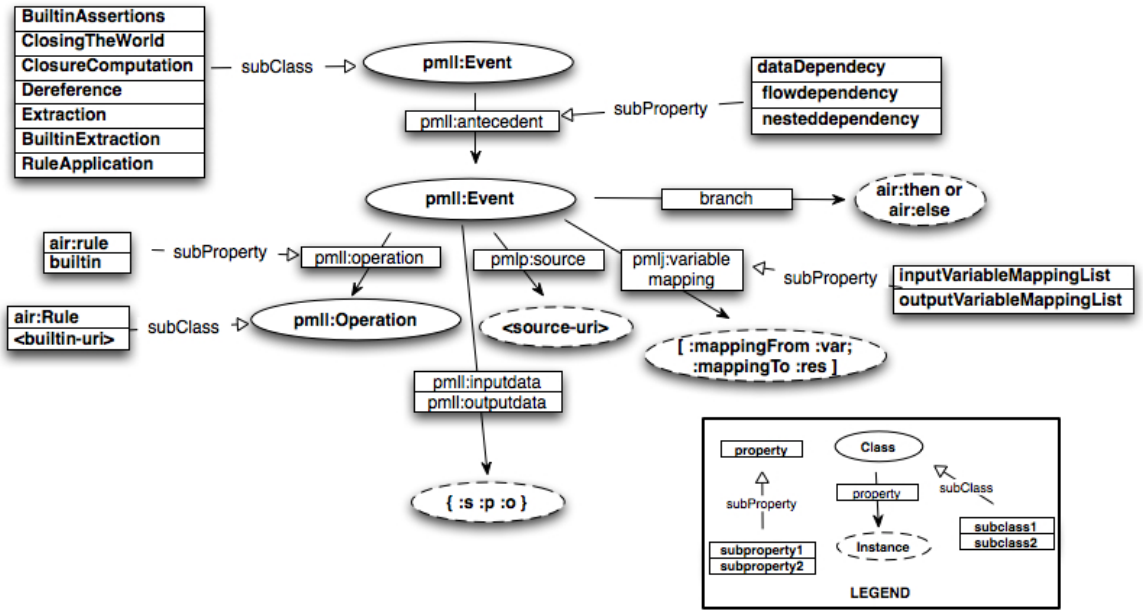

Figure 8: AIR Justification Ontology: All concepts defined without a prefix are in the airj namespace

soning steps. The AIR justification ontology consists of three main classes — ***pmll:Event***, ***pmlj:Mapping*** and ***pmll:Operation***. The Event class may be categorized into BuiltinAssertion, BuiltinExtraction, ClosingTheWorld, ClosureComputation, Dereference, Extraction, and RuleApplication events depending on the operation performed.

The AIR reasoner computes the closure of input facts with respect to a given AIR ruleset. An ***airj:ClosureComputation*** event captures this operation. The events of all other types are all part of some ClosureComputation event.

The input to an AIR ruleset may be contained in N3 or RDF documents on the Web or on a local machine. They are dereferenced to retrieve their graph representation. Each dereference is encoded as an ***airj:Dereference*** event. In the example in Figure 9, the document located at <`http://dig.csail.mit.edu/2011/Papers/WWW-AIR/example/req`> is dereferenced to get the RDF graph `_:g0` in event `_:d0`, from which it may be determined whether Carol's friend Joe is allowed to access an image.

The ***airj:RuleApplication*** events `_:ra0` and `_:ra1` in Figure 9 represent rule firing events. They are linked to the rule that fired by the ***air:rule*** property. The RuleApplication event for a nested rule has a special flow dependency, or ***airj:nestedDependency***, on the RuleApplication event where this rule was activated. `_:ra1` features such a nestedDependency on `_:ra0`, and the input variable bindings of `_:ra1` are the same as the output variable bindings of the `_:ra0`. The input and new variable bindings after a rule fires are declared using the ***airj:outputVariableMappingList*** property. Any asserted triples are declared using the ***pmll:outputdata*** property, and the event may be annotated with a natural language description specified by the ***air:description*** property. The ***air:branch*** property states whether the rule succeeded or failed at matching.

Note that RuleApplication events may have data dependencies on other events. The condition can be satisfied by triples from more than one input log, or by triples asserted in prior RuleApplication events. They may also have flow control dependencies on prior ***airj:ClosingTheWorld*** events.

A ClosingTheWorld event, such as that in Figure 10, refers to a temporary closure of the world before failed rules are fired. A ClosingTheWorld event has data and/or flow control dependencies on all prior RuleApplication events. These dependencies are represented through ***airj:dataDependency*** and ***airj:flowDependency*** properties, respectively.

In Figure 10, an alternate justification beginning similarly to that in Figure 9, a ClosingTheWorld event, `_:ctw0`, is used to close the world before deciding `:ScopedExecutionOfCarolRule` so that it may follow the ***air:else*** action. This allows the reasoner to conclude that the requester, Joe, is *not* allowed to access the image given Bob's ruleset (and Carol's rule). Note that the event `_:ctw0` has a dataDependency on all previous BuiltinExtraction and Dereference events which built the now-closed knowledge base.

Apart from the input triples, there are N3Logic triples that are tautologically true. Some of these are built-in assertions. In practice, built-in triples are created dynamically. However, we abstract this process and represent it through a series of ***airj:BuiltinAssertion*** and ***airj:BuiltinExtraction*** events. The output of a BuiltinAssertion event is assumed to be the graph that contains all the true assertions (potentially unbounded in number) for the function specified by the ***airj:builtin*** property. The assertions needed to match rule conditions are then extracted from this output in a BuiltinExtraction event. The ***airj:Extraction*** event from which BuiltinExtraction is derived, is used to encode the step where a subgraph is obtained from a graph. The input to BuiltinExtraction event is implicit, and may be determined by the BuiltinAssertion event on which the extraction has a dataDependency.

```
_:d0 a airj:Dereference ;
  airj:source
    <http://dig.csail.mit.edu/2011/Papers/WWW-AIR/example/req#> ;
  pmll:outputdata _:g0 .
# similar events for carol's & joe's foafs & carol's policy

_:in a airj:BuiltinAssertion ;
  airj:builtin log:includes .
# similar events for air:justifies & math:greaterThan

_:be0 a airj:BuiltinExtraction ;
  airj:dataDependency _:in ;
  pmll:outputdata { _:g1
   log:includes {
      <http://dig.csail.mit.edu/2008/02/rmp/carol-foaf#i>
        foaf:knows
        <http://dig.csail.mit.edu/2008/02/rmp/joe-foaf#me> } } .
# similar events for air:justifies & log:includes

_:be3 a airj:BuiltinExtraction ;
  #dataDependency on BuiltinExtraction for math:greaterThan
  pmll:outputdata { 100 math:greaterThan 70 } .

_:mapping0 a pmlj:Mapping ;
  airj:mappingFrom :REQUESTER ;
  airj:mappingTo
    <http://dig.csail.mit.edu/2008/02/rmp/joe-foaf#me> .
# Similarly Mapping _:mapping1 for :PIC

_:ra1 a airj:RuleApplication ;
  air:rule :ScopedExecutionOfCarolRule ;
  airj:nestedDependency _:ra0 ;
  airj:branch air:then ;
  airj:dataDependency _:d0, _:d2, _:d3, _:be1, _:be2, :be3 ;
  airj:outputVariableMappingList (_:mapping0 _:mapping1) ;
  pmll:outputdata {
    <http://dig.csail.mit.edu/2008/02/rmp/joe-foaf#me>
       req:compliant-with :BobRuleSet} ;
  air:description ("Carol trusts the requester, "
      <http://dig.csail.mit.edu/2008/02/rmp/joe-foaf#me>
      ", more than 70 percent, so I will allow access") .

_:ra0 a airj:RuleApplication ;
  air:rule :ViewImageRule2 ;
  #... other properties similar to :ra1
```

Figure 9: An AIR Justification: Part of the justification allowing access to the picture to Joe, Carol's friend, based on the rules in Figure 6. Note that all events generated have been assigned to blank nodes (e.g. `_:d0`), but this is not a requirement.

In the example in Figure 9, we require that the graph `_:g1`, representing Carol's foaf file, include the triple <**http://dig.csail.mit.edu/carol-foaf#i**> **foaf:knows** <**http://dig.csail.mit.edu/joe-foaf#me**>. This is tested using the ***log:includes*** built-in. Using the abstract event `_:in`, we get all ***log:includes*** triples that can be true, and extract the one relevant for reasoning in the `_:be0` event. Thus, `_:be0` has a data dependency on `_:in`.

By manually or automatically tracing the output data and rule dependencies from each BuiltinExtraction, Dereference, or RuleApplication event, we may determine the order in which rules were applied and what data was used. This also allows for the construction of meaningful, human-readable justification traces and interfaces [9].

5.2 Justification Generation

AIR supports justification generation for every action taken by the reasoner. The reasoner annotates every action — firing of rules, execution of built-ins, and closing of the world — with relevant information. Furthermore, it

```
_:ctw0 a airj:ClosingTheWorld ;
   airj:dataDependency _:be0 , _:d0 ;
   airj:flowDependency _:ra0 ;
   # And others in the same way.

_:ra1 a airj:RuleApplication ;
   air:rule :ScopedExecutionOfCarolRule ;
   airj:branch air:else ;
   airj:nestedDependency _:ra0 ;
   # ...
   airj:dataDependency _:ctw0 ;
   airj:outputdata {
     <http://dig.csail.mit.edu/2008/02/rmp/joe-foaf#me>
       req:non-compliant-with :BobRuleSet . } .
```

Figure 10: Justifying Joe's rejection: Closing the world to assert *air:else*.

tracks dependencies between the actions and generates the justification as described in [9]. The default justification for a conclusion is constructed by recursively taking the union of descriptions of dependent actions starting with the action that asserted the conclusion.

Though knowing the rules and facts from which a conclusion is derived is useful, it does not describe what the rule was attempting to do. In order to provide natural-language explanations, we allow ***air:descriptions*** to be added to ***air:actions***. These descriptions are English sentences and can contain variable values. The ***air:description*** property is a list instance, where list items are enclosed in brackets and separated by commas. Each list item can either be a string enclosed in quotes or a quantified URI variable. During the reasoning process, each variable is replaced by its current value and inserted into the description string. For example, the ***description*** of `:ScopedExecutionOfCarolRule` from Figure 6 is a list consisting of one variable, `:REQUESTER`, and two strings — *Carol trusts the requester* and *more than 70 percent, so I will allow access.*

Sometimes, the default justification can be unwieldy or very revealing, and must be modified to hide trivial or sensitive information. AIR provides mechanisms to *declaratively modify justifications*. Rules in AIR may be declared to be ***air:Hidden-rules*** or ***air:Elided-rules*** to suppress justifications for certain actions. The detailed justifications for actions executed when an ***Elided-rule*** fires are suppressed, and only the natural-language description is provided. The justification for actions executed when a ***Hidden-rule*** or its descendants fire are suppressed completely. This flexibility to control the level of details, at a rule-based granularity, helps rule authors to adjust the justification so that sensitive information is not revealed and so explanations are not overly verbose. Nesting of rules can be used to split a rule's conditions across multiple rules when parts of the graph pattern refer to sensitive (or insignificant) information, so that such rules may be elided or hidden.

From our use case, Bob's policy for Carol's friends is to use Carol's policy to decide whether or not he trusts them enough to show a picture to them. However, Carol may not be comfortable sharing her trust valuations for friends. For instance in the justification in Figure 9, the trust value for Joe, is revealed in `_:be3` to be greater than 70 . Bob may choose to hide these details. In this case, `:ScopedExecutionOfCarolRule` from Figure 6 could be declared to be ***a air:Elided-rule*** instead of ***a air:Belief-rule***. As illustrated in Figure 11, the justification for Joe's permission

```
...
@forSome _:ra1 .
_:ra1 a airj:RuleApplication ;
  airj:nestedDependency _:ra0 ;
  pmll:outputdata {
    <http://dig.csail.mit.edu/2008/02/rmp/joe-foaf#me>
       req:compliant-with :BobRuleSet} ;
  air:description ("Carol trusts the requester, "
      <http://dig.csail.mit.edu/2008/02/rmp/joe-foaf#me>
      ", more than 70 percent, so I will allow access") .
...
```

Figure 11: Justification of Elided Rule: Justification for permission to Carol's friend does not contain the *Elided-rule* `:ScopedExecutionOfCarolRule`

would not contain information about `:ScopedExecutionOfCarolRule` and the trust valuation would not be revealed.

6. RELATED WORK

There are many rule languages and rule systems, and Liang et. al give a nice overview of popular, and often advanced, rule systems [16]. Examples of Web rule languages include N3Logic [4], Networked Graphs (NG) [22] and SWRL[10], and some rule systems are Jena[11], Jess[12], Ontobroker [13], SILK [7] and XSB [5]. While AIR, N3Logic, NG, SWRL and Jena are rule languages/ systems for triples, others are more generic reasoners which can be used for the semantic web data as well. A detailed comparison on semantics and expressiveness of AIR with other systems/ languages is provided in [13].

In AIR, rules can have global unique identifiers, and they can be reused by direct reference. This feature is unique to AIR. Likewise only AIR supports rule nesting. AIR and other systems can retrieve remote Semantic Web data, however in AIR rule conditions can also be matched against remote SPARQL end-points using the ***sparql:queryEndpoint*** builtin. While most of the systems can provide a trace for how the inference was drawn, none support customized explanations such as in AIR.

In the past there has been some research in generating customized explanations using proof traces. The Ontonova system [1] provides natural-language explanation of proof trees for conclusions using Ontobroker and meta-inferencing. Meta-inferencing rules are defined for different rule instantiations and applied over the logs generated by Ontobroker during the inferencing process.

The defeasible reasoning system, DR-DEVICE, has a similar mechanism for generating proofs[2]. Defeasible rules can be translated to XSB rules, and interpreted by XSB. The XSB trace is processed and tagged with a proof schema. In the context of OWL, a subset of the ontology sufficient for OWL entailment to hold is treated as the justification for the entailment and algorithms exist to derive the minimal such subset [8, 12].

In contrast to the above approaches, AIR allows rule authors to specify the natural language explanation of the rule within the rule definition itself, and allows the details revealed in the justification to be selectively pruned.

The derivation steps leading to a conclusion can be explained through the Inference Web (IW) explanations [18]. IW explanations are proofs marked up in PML and are portable. For this the reasoner and inference steps must be registered with the IWBase. We are investigating the registration of AIR inference steps with IWBase. However, we can not hide certain derivation steps using IW as required by hidden and elided AIR rules.

AIR supports scoped contextualized reasoning through ***log:includes***, ***log:notIncludes*** and ***air:justifies*** builtins. Scoped reasoning is important for Web rule languages because information on the Web is often assumed to be incomplete or inconsistent, and its correctness is subject to the trustworthiness of the source. Other than AIR and N3Logic, NG, Ontobroker and SILK support contextualized reasoning. The scope or context, defined by some data and/ or rules, can be chosen dynamically in AIR, unlike any other system, to the best of our knowledge.

The Rule Interchange Format (RIF) [14] is an effort towards developing standards to facilitate interoperability of rules in these disparate rule based systems over the Web. RIF has two major dialects — Framework for Logic Dialects and Production Rule Dialect. The semantic differences between each of these and AIR is discussed in [13]. In addition, while RIF is meant for general rule interchange, AIR is explicitly designed to operate on Semantic Web data. Furthermore, the representation of RIF in RDF currently exists only as a W3C working draft.[14] RIF documents are able to import other RIF documents, but, unlike AIR, rules themselves cannot be interlinked. One of the advantages of interlinking rules is that we can add additional conditions to rules before reuse, i.e. defining specialization of rules. Furthermore, to the best of our knowledge there is no explicit notion of scoped reasoning in RIF.

The compatibility of rules with semantic web data in various languages RDF, RDFS, OWL 2 DL and OWL 2 Full is clearly defined for RIF [15], but remains future work for us to do in AIR. One of the motivating use cases for defining such compatibilities is to help rule publishers to extend OWL ontologies with rules. Like in RIF, OWL 2 RL [16] reasoning can be implemented in AIR, and therefore ontologies with OWL 2 RL semantics can be used with additional AIR rules.

7. SUMMARY AND FUTURE WORK

We have found that AIR's expressiveness and functionality allow it to easily capture real world rules and policies while leveraging (Semantic) Web data and protocols. It supports ***Linked Rules*** so that rules can be developed and reused in a manner similar to Linked Data, provides functions for scoped contextualized reasoning, provides justification for its inferences which can be used to evaluate the quality and trustworthiness of results. Though our use case demonstrates how AIR can be used in collaborative environments, AIR has also been used for information sharing for access control and policy management, to secure SPARQL endpoints, to check the compliance of queries against privacy policies and as an accountability mechanism for checking whether audit logs comply with usage restriction policies.

[10]http://www.daml.org/rules/proposal/
[11]http://jena.sourceforge.net/
[12]http://www.jessrules.com/
[13]http://www.ontoprise.de/en/home/products/ontobroker/
[14]http://www.w3.org/TR/rif-in-rdf/
[15]RIF RDF and OWL compatibility- http://www.w3.org/TR/2009/WD-rif-rdf-owl-20090703/
[16]http://www.w3.org/TR/owl2-profiles/#OWL_2_RL

Thus far, our focus has been on studying the rule requirements of open Web information systems, designing appropriate features in the rule language, and implementing AIR-based systems. Moving forward we will work on handling conflicts between rules and on enabling default behavior when none of the conditions of a ***RuleSet*** match. We are also interested in studying the performance and scalability of AIR and plan to use or extend existing benchmarks [16]. Furthermore, we will define an AIR to RIF translation so that AIR rules can be exchanged with RIF-aware systems.

8. REFERENCES

[1] J. Angele, E. Moench, S. Staab, and D. Wenke. Ontology-based Query and Answering in Chemistry: Ontonova @ Project Halo. In *Proceedings of the Second International Semantic Web Conference (ISWC2003). 2003*, pages 913–928. Springer Verlag, 2003.

[2] N. Bassiliades, G. Antoniou, and G. Governatori. Proof explanation in the DR-DEVICE system. In *Proceedings of the 1st International Conference on Web Reasoning and Rule Systems*, pages 249–258, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 81–95, May 2005.

[4] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler. N3Logic: A Logical Framework For the World Wide Web. *Journal of Theory and Practice of Logic Programming*, 2007.

[5] W. Chen and D. S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, 1996.

[6] M. Cherian, L. Kagal, and E. Prud'hommeaux. Policy Mediation to Enable Collaborative Use of Sensitive Data. In *The Future of the Web for Collaborative Science Workshop at WWW2010*, April 2010.

[7] B. Grosof, M. Dean, and M. Kifer. The SILK System: Scalable and Expressive Semantic Rules. In *8th International Semantic Web Conference (ISWC2009)*, October 2009.

[8] M. Horridge, B. Parsia, and U. Sattler. Laconic and Precise Justifications in OWL. In *Proceedings of International Semantic Web Conference*, pages 323–338, 2008.

[9] L. Kagal, C. Hanson, and D. Weitzner. Using Dependency Tracking to Provide Explanations for Policy Management. In *IEEE International Symposium on Policies for Distributed Systems and Networks*, 2008.

[10] L. Kagal, I. Jacobi, and A. Khandelwal. From Truth Maintenance to Trust Management on the Semantic Web. Technical report, MIT Technical Report, 2010.

[11] L. Kagal and J. Pato. Preserving Privacy Based on Semantic Policy Tools. *IEEE Security and Privacy Sp Issue on Privacy Preserving Sharing of Sensitive Information (PPSSI)*, 2010.

[12] A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, pages 267–280, Berlin, Heidelberg, 2007. Springer-Verlag.

[13] A. Khandelwal, J. Bao, L. Kagal, I. Jacobi, L. Ding, and J. Hendler. Analyzing the AIR Language: A Semantic Web (Production) Rule Language. In P. Hitzler and T. Lukasiewicz, editors, *Web Reasoning and Rule Systems*, volume 6333 of *Lecture Notes in Computer Science*, pages 58–72. Springer Berlin / Heidelberg, 2010.

[14] M. Kifer. Rule Interchange Format: The Framework. In *Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems*, pages 1–11, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] V. Kolovski, Y. Katz, J. Hendler, D. Weitzner, and T. Berners-Lee. Towards a Policy-Aware Web. In *Semantic Web and Policy Workshop at the 4th International Semantic Web Conference*, 2005.

[16] S. Liang, P. Fodor, H. Wan, and M. Kifer. OpenRuleBench: An Analysis of the Performance of Rule Engines. In *18th International World Wide Web Conference (WWW2009)*, April 2009.

[17] C. man AuYeung, L. Kagal, N. Gibbins, and N. Shadbolt. Providing Access Control to Online Photo Albums Based On Tags and Linked Data. In *AAAI Spring Symposium on Social Semantic Web: Where Web 2.0 Meets Web 3.0*, March 2009.

[18] D. L. Mcguinness and P. P. da Silva. Explaining answers from the Semantic Web: the Inference Web approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):397–413, October 2004.

[19] D. L. McGuinness, L. Ding, P. P. da Silva, and C. Chang. PML 2: A Modular Explanation Interlingua. In *AAAI 2007 Workshop on Explanation-aware Computing*, 2007.

[20] A. Polleres, C. Feier, and A. Harth. Rules with Contextually Scoped Negation. In *Proceedings of 3rd European Semantic Web Conference (ESWC2006*, pages 332–347. Springer, 2006.

[21] T. C. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. *Foundations of deductive databases and logic programming*, pages 193–216, 1988.

[22] S. Schenk and S. Staab. Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In *Proceeding of the 17th international conference on World Wide Web*, pages 585–594, New York, NY, USA, 2008. ACM.

[23] K. K. Waterman and S. Wang. Prototyping fusion center information sharing; implementing policy reasoning over cross-jurisdictional data transactions occurring in a decentralized environment. In *IEEE Conference on Homeland Security Technologies (IEEE HST 2010)*, November 2010.

[24] D. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. Sussman. Information Accountability. *Communications of the ACM*, pages 82–87, June 2008.